Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Team Software Process for Secure Systems Development

James W. Over
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Sponsored by the U.S. Department of Defense

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
MAR 2002		**00-00-2002 to 00-00-2002**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Team Software Process for Secure Systems Development	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,Software Engineering Institute,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**35**	

Carnegie Mellon
Software Engineering Institute

Trademarks and Service Marks

The following are service marks of Carnegie Mellon University.

- CMM Integration℠
- CMMI℠
- Team Software Process℠
- TSP℠
- Personal Software Process℠
- PSP℠

The following are registered trademarks of Carnegie Mellon University.

- Capability Maturity Model®
- CMM®

Carnegie Mellon
Software Engineering Institute

Overview

- Defective software is not secure
- The Team Software Process
- TSP and secure systems development

Carnegie Mellon
Software Engineering Institute

Defective Software is not Secure

Common software defects are a principal cause of software security incidents.

- Over 90% of software security incidents are due to attackers exploiting known software defect types.[1]
- Analysis of forty-five e-business applications showed that 70% of the security defects were design defects.[2]

Conclusion: there is no such thing as a poor quality secure system.

- CERT/CC
- "The Security of Applications: Not All Are Created Equal", by Andrew Jacquith.

Security Design Defects

Examples

- Failure to authorize and authenticate users
- Failure to validate user input
- Failure to encrypt and/or protect sensitive data

Everyday software “bugs” are also a major risk.

For example, a buffer overflow can cause system failure, or allow a hacker to take control of your system.

Many common defect types can produce a buffer overflow.

- declaration error
- logic errors in loop control or conditional expression
- failure to validate input
- interface specification error

Carnegie Mellon
Software Engineering Institute

The Software Quality Problem

Software quality is highly variable and generally poor in non-mission critical systems.

Widely-used operating systems and applications software are known to have more than 2 defects per KSLOC, or 2000+ defects per million SLOC.

If only 5% of these defects are potential security concerns, there are 100 vulnerabilities per million SLOC.

Software Practice and Quality

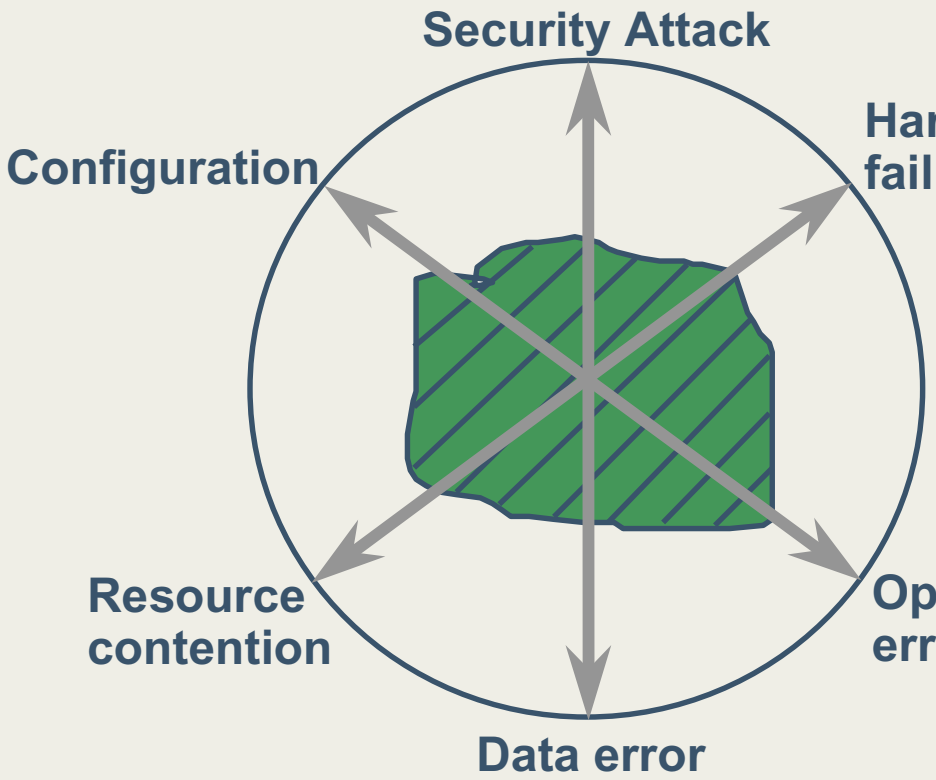
Software is the only modern technology that ignores quality until test. Typically, software engineers

- do not plan their own work
- race through requirements and design
- do the design while coding

These practices introduce volumes of defects.

- Experienced engineers inject a defect every 7 to 10 lines of code.
- For even moderate-sized systems, this amounts to thousands of defects.
- Most of these defects must be found in test.
- This usually takes about half of the development schedule.

The Problem with Testing



Safe region = tested (shaded)

Unsafe region = untested (unshaded)

Principles of Software Engineering

Current software practice violates well understood principles of software engineering.

Examples of software engineering principles

- the need for accurate plans
- the importance of detailed, verifiable designs
- early defect removal
- effective inspections
- focus on quality throughout

Why are these principles not applied?

Principles Are Not Enough

Principles are easy to understand, but harder to follow.

To apply these principles in practice you also need

- a supportive infrastructure and environment
- an operational process (rules and steps) to put the principles into practice
- a measurement system to manage and control the result

Software engineers also need to be convinced of the benefits of disciplined software engineering methods.

Design Principles for Secure Applications

The principles for producing secure applications are also well known and easy to understand.

Examples

- Authorize and authenticate all users
- Mistrust all user input
- Encrypt sensitive data from login to logout
- Protect persistent data

What is the Issue?

Lack of understanding is not the issue.

A lack of data is not the issue. A vast amount of incident-specific and system-specific data exists.

Training is not the issue. Training is available for

- writing secure applications
- network administration

Support is not the issue. There are emergency response centers (including the CERT/CC), guidelines, checklists, best practices, etc.

More Is Needed

With all the information and resources available, why is security still an issue?

Maybe there is a need for more.

- an environment that fosters good practice
- operational processes based on engineering principles
- disciplined practitioners that adhere to these principles
- predictive measures

Carnegie Mellon
Software Engineering Institute

Overview

Defective software is not secure

The Team Software Process

TSP and secure systems development

The Team Software Process -1

The Team Software Process (TSP) is an operational process designed to support well-established principles of software engineering.

The principal objectives of the TSP are

- help software engineering teams build quality products within cost and schedule constraints
- build teams quickly and reliably
- optimize team performance throughout a project

The Team Software Process -2

TSP incorporates best practices of software engineering in a single integrated package, e.g.

- team project management
- product quality management
- process management
- risk management
- software metrics

With TSP, software teams

- build detailed, accurate plans
- manage and track their commitments to within +/-10%
- produce near defect-free software with typically less than 0.1 defects/KSLOC

Personal Software Process

To use the TSP, software developers must first be trained in the Personal Software Process (PSP).

The PSP provides software developers with the skills and self-convincing evidence of the benefits of software engineering practice.

In using the PSP, software developers

- follow a defined and measured personal process
- plan every job before they do it
- gather time, size, and defect data as they work
- use these data to manage their personal work and ensure the quality of the products they produce

Software Quality with TSP

The quality management practices in TSP and PSP dramatically reduce the product defect content.

The following results are drawn from

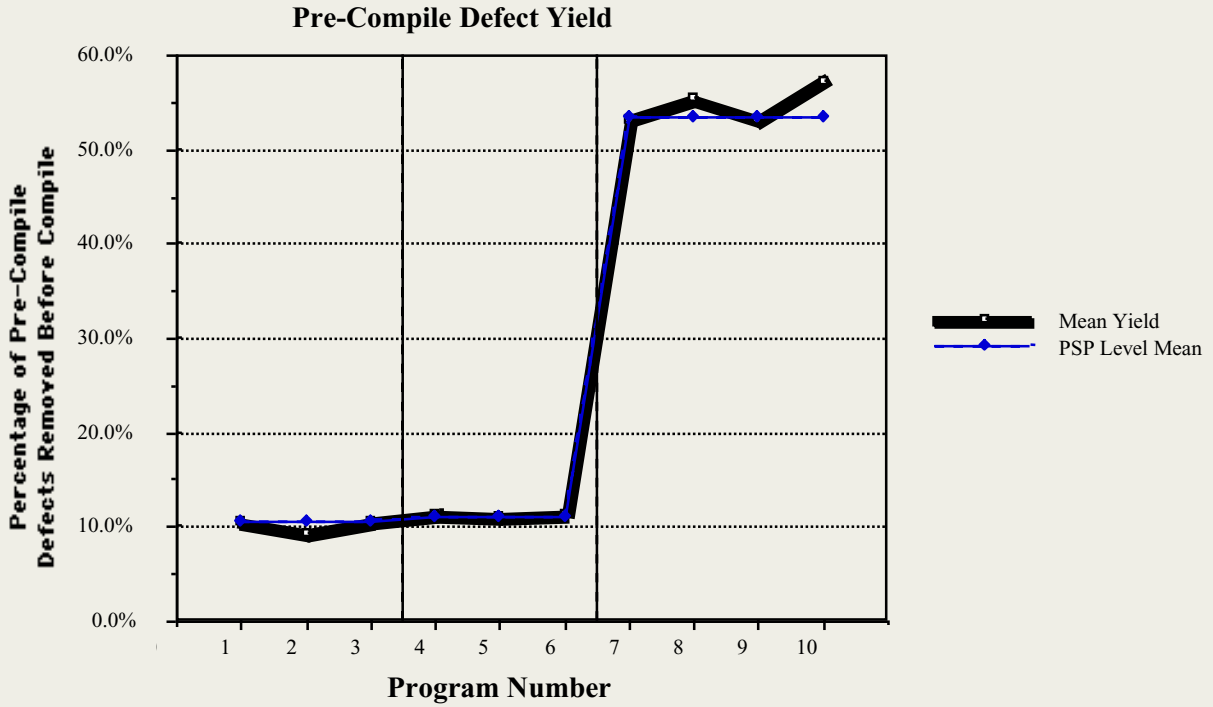
- PSP training data on 298 software developers
- TSP data from 18 projects in four organizations

The results show the effect of TSP and PSP on

- process quality
- design time
- product quality
- system test duration

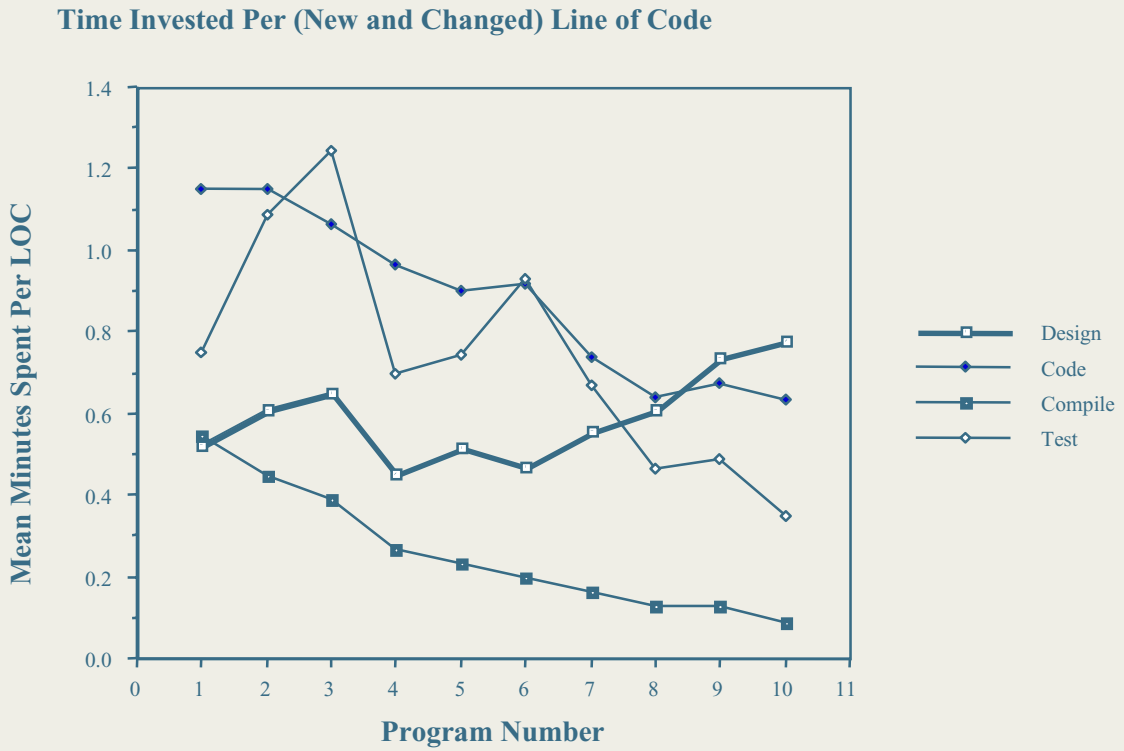
Carnegie Mellon
Software Engineering Institute

Process Quality Results



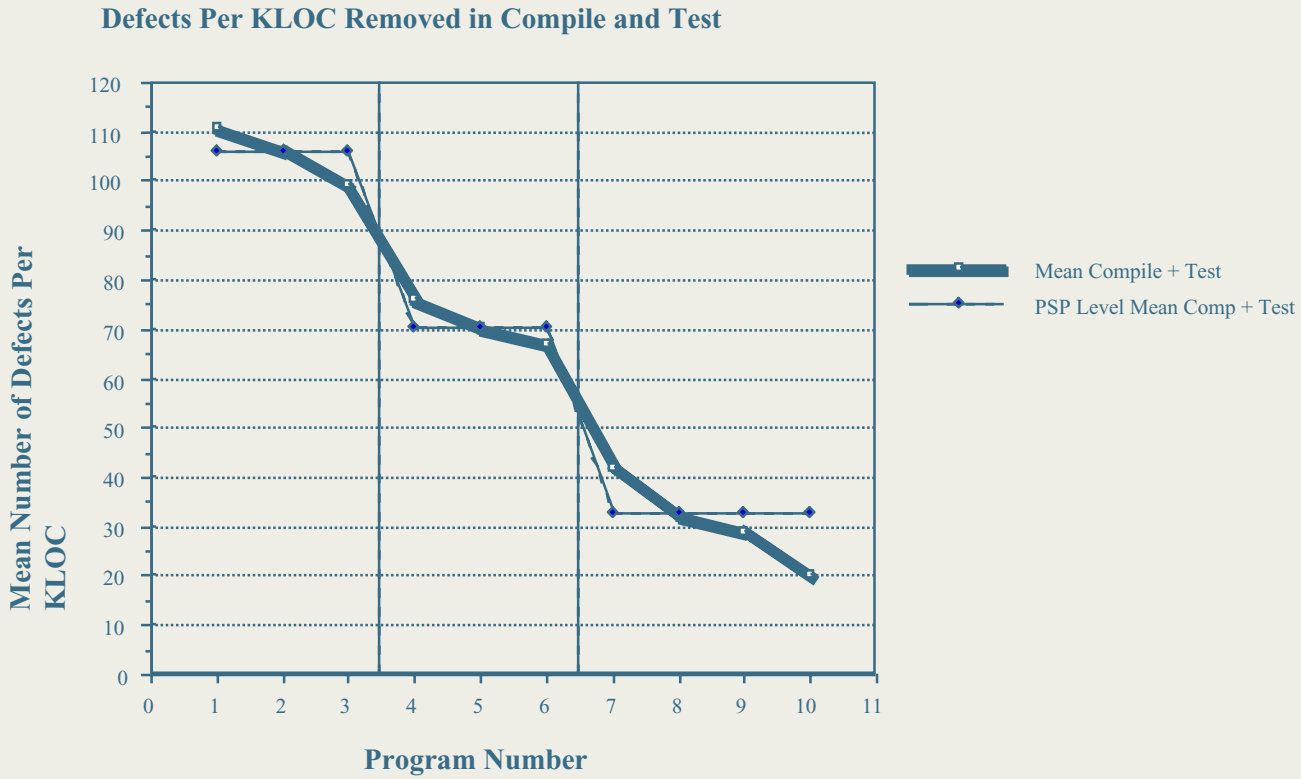
Carnegie Mellon
Software Engineering Institute

PSP Design Time Results

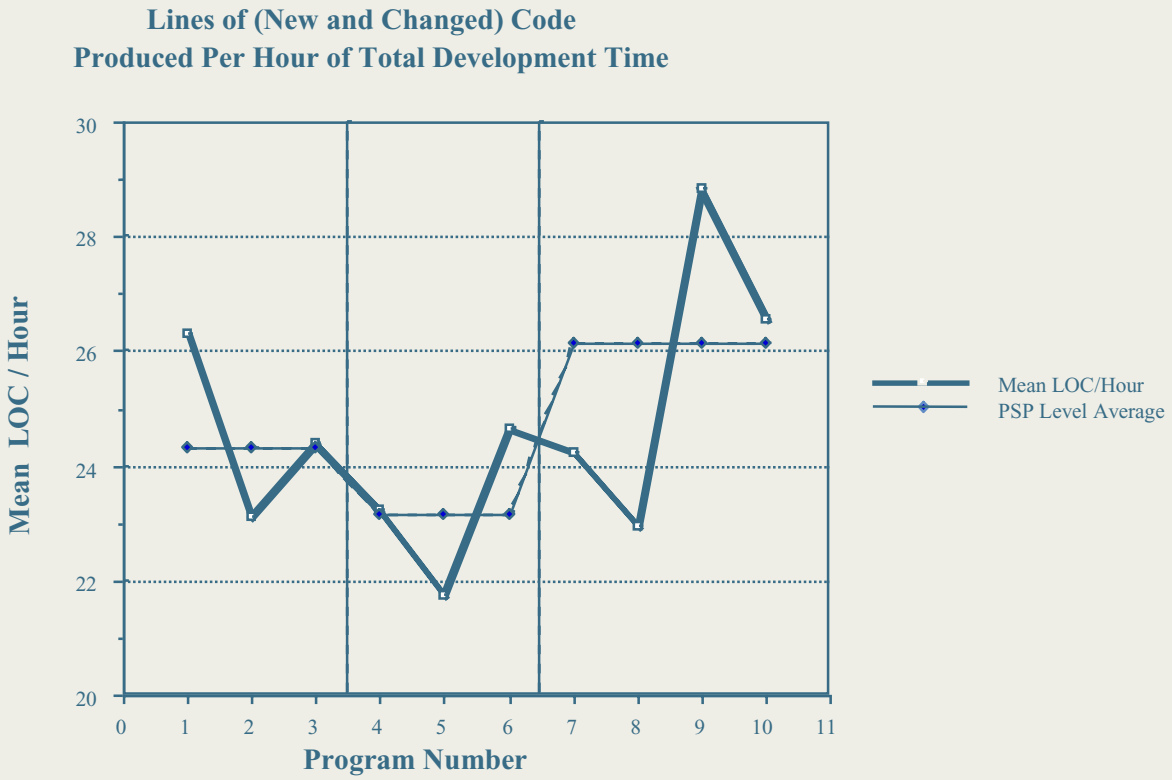


Carnegie Mellon
Software Engineering Institute

PSP Product Quality

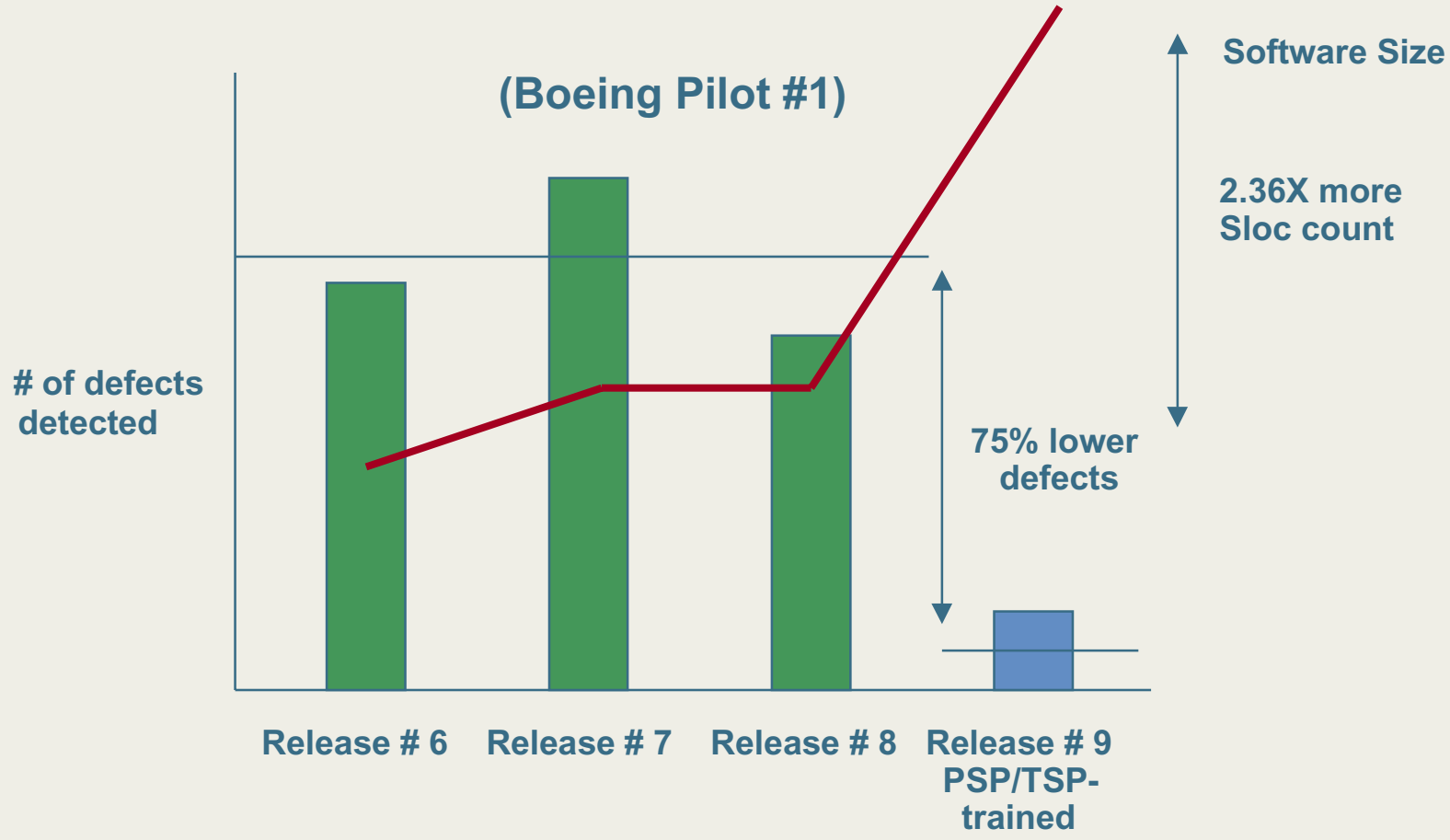


PSP Productivity Results

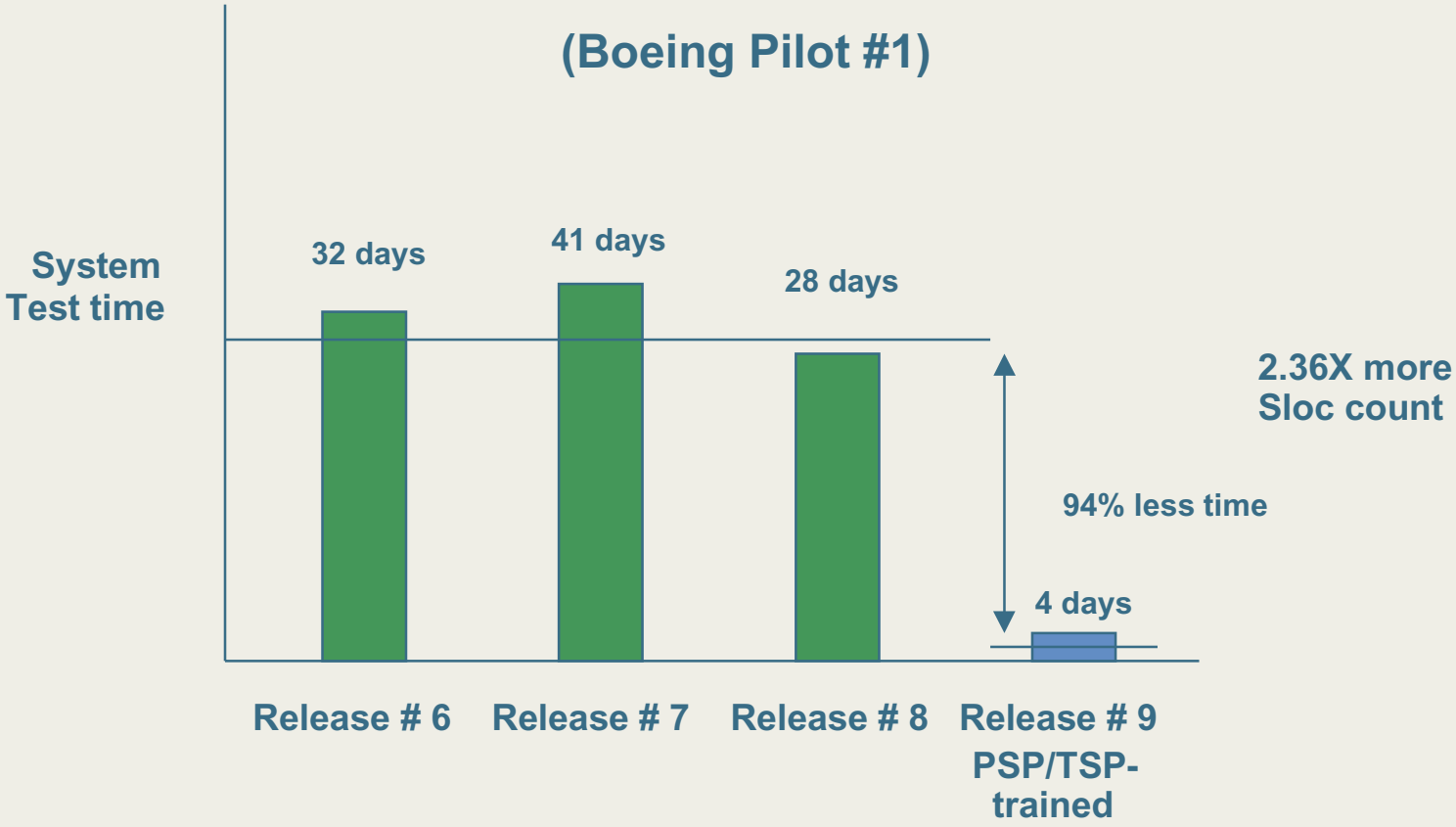


Carnegie Mellon
Software Engineering Institute

TSP Quality Improvement -1



TSP Quality Improvement -2



Carnegie Mellon
Software Engineering Institute

TSP Project Results

	Plan	Actual
Size Estimate	**110,000 LOC**	**89,995 LOC**
Effort Estimate	**16,000 hours**	**14,711 hours**
Schedule	**77 weeks**	**71 weeks**

Product Quality **(Defects/KLOC removed in phase)**

• Integration	**1.0**	**0.2**
• System Test	**0.1**	**0.4**
• Field Trial	**0.0**	**0.02**

Benefits

- Quality levels improved 20 times over prior projects.
- Actual effort and schedule were within 8% of plan (early).

Carnegie Mellon
Software Engineering Institute

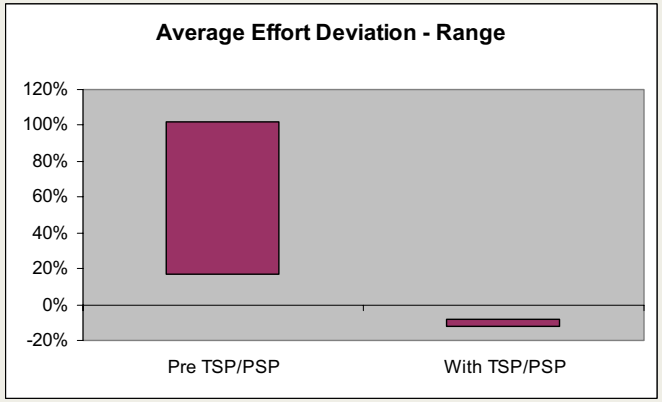
TSP Results Summary -1

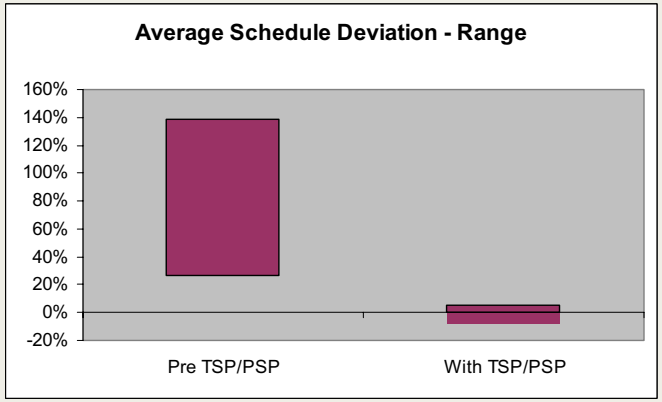
Category	Without TSP	With TSP
Average schedule deviation - range	27% to 112%	-8% to 5%
Average effort deviation - range	17% to 85%	-8% to -4%
Acceptance test product quality (defects/KLOC)	0.1* to 0.7	0.02 to 0.1
System test savings (cost to system test 1000 LOC)	1 to 5 days	0.1 to 1 days
Number of post-release defects per KLOC	0.2 to 1+	0 to 0.1

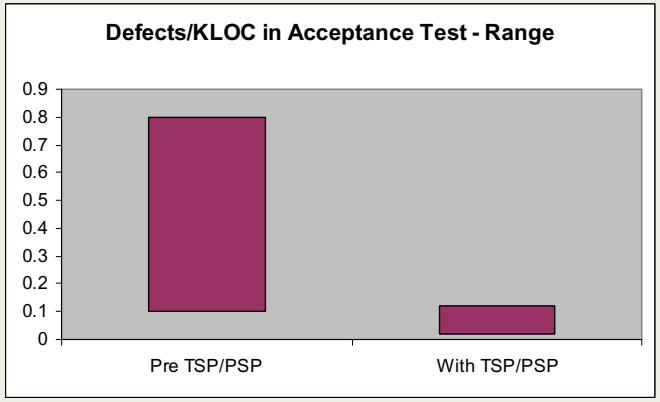
* This data (.1 defects/KLOC in acceptance test) is from a CMM Level 5 organization

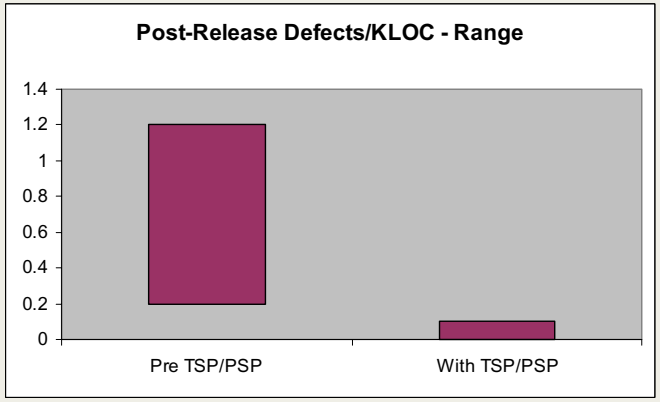
Carnegie Mellon
Software Engineering Institute

TSP Results Summary -2









Carnegie Mellon
Software Engineering Institute

Overview

Defective software is not secure

The Team Software Process

TSP and secure systems development

TSP and Secure Systems -1

The TSP provides a framework, a set of processes, and disciplined methods for producing quality software.

Software produced with TSP has one or two orders of magnitude fewer defects than current practice.

- 0.02 defects/KSLOC vs. 2 defects/KSLOC
- 20 defects per MSLOC vs. 2000 defects per MSLOC

If 5% of the defects are potential security holes, with TSP there would be 1 vulnerability per million SLOC.

TSP and Secure Systems -2

TSP also address the need for

- professional behavior
- supportive environment
- sound software engineering practice
- operational processes
- software metrics

With tailoring, TSP could be even more effective in this development domain.

Carnegie Mellon
Software Engineering Institute

TSP for Secure Systems -1

TSP for Secure Systems is an applied research effort to enhance TSP for the secure systems domain.

Using design principles for secure applications, the TSP could be extended to incorporate

- secure design process
- secure implementation process
- secure review and inspection process
- secure test process
- security-related predictive measures

Carnegie Mellon
Software Engineering Institute

TSP for Secure Systems -2

The goal of this effort is to develop a process that

- supports secure systems development practices
- predicts the likelihood of latent security defects
- can be dynamically tailored to respond to new threats

The TSP for secure systems project is planned for FY03 and will be a collaborative effort involving

- industry and government partners
- SEI/NSS program
- SEI/TSP initiative

Carnegie Mellon
Software Engineering Institute

TSP Secure Systems Pilot

A key element of the TSP for Secure Systems project will be pilot projects.

The pilot project will help in developing the specific technical solutions.

- design and implementation practices
- review methods and checklists
- tools

We are currently seeking organizations that are interested in participating.

Summary

TSP helps organizations establish a mature and disciplined software engineering practice that

- improves cost and schedule predictability
- reduces time to market
- produces high-quality, reliable software, with fewer security-related defects

The TSP for secure systems effort will build on these capabilities to create a mature process, with specific features for building secure systems.

Carnegie Mellon
Software Engineering Institute

For More Information

Visit the Software Engineering Institute web site at
www.sei.cmu.edu

Visit the TSP web site at
www.sei.cmu.edu/tsp

Contact SEI Customer Relations at 412-268-5800